

**REMARKS**

Claims 1-27 are presented for prosecution. Claims 9 and 11 are amended. Claims 25-27 are new.

Claims 1-24 rejected as being unpatentable in view of U.S. Pat. 6,571,389 to Spyker et al., U.S. Pat. 6,854,006 to Giroir et al., and U.S. Pat. 6,535,804 to Schmidt et al.

Many of the Office Action rejections appear to be based on a misunderstanding of the present invention. As is explained in the specification of the present application, web browsers are typically used to interface with network servers, especially on the internet. In order to improve portability between computer platforms, many software routines provided by network servers are written using a Java© based approach. A web browser includes a Java interpreter to accommodate these Java based routines. However, not all web browsers interpret Java routines similarly, or include the latest version of a Java interpreter, or permit the Java routine much freedom for execution. This is because web browsers are limited not only by their innate ability to run a Java routine, but also by network security protocols and limitations.

Often, Java routines implemented via a network on a web browser are implemented as Java "applets". The web browser treats the applet like any other software object, i.e. like an image, text box, etc. Although the applet may be displayed in a window separate from the web browser, the applet is still a part of the web browser (i.e. subject to the security limitations of the web browser), and is thus limited by the web browser. However, because the applet is run within the web browser, the applet can receive execution parameters directly from the web browser.

One of the objectives of the present invention is to remove the limitations a web browser places on the execution of a Java routine without eliminating the ability of the web browser to continue to transfer parameters to the Java routine. The present invention achieves this by implementing the Java routines as an independent (i.e. stand-alone) Java application.

There is a distinct difference between an applet and an application. As stated above, the applet is an integral part of (and thus not independent of) the web browser. As result, the applet is subject to the same limitations as the web browser even if the applet is executed in a separate window, and appears independent from the web browser. By contrast, the application is a stand-alone unit, subject to a different Java engine that can be maintained up-to-date without affecting any web browser, which is a separate stand-alone application.

Thus, one of the issues addressed by the present application is how to create a stand-alone Java application that is not only launched by a web browser, but continues to receive parameters from the web browser. That is, how does one make an application that looks and feels like applet under control of a web browser without being subject to the web browser's limitations. The present invention accomplishes this by means of the ports the host computer (i.e. network client) uses to link its web browser to the network server.

At this point, it may be advantageous to briefly explain what is meant by "port", as used in the present application. As is known in the art, network communication among computers is managed by means of network "ports" within each computer. It should be emphasized that these ports are not mechanical connection couplers, but rather memory locations used by network software to keep track of information packets transferred between a multitude of different applications between many different machines on a network. This use of the term "port", as used in network communications is explained in text book, "Java, How to Program", Third Edition by Deitel and Deitel (Exhibit A), page 945, third and fourth paragraphs, wherein it states,

"... [Note: Ports in this case are not physical hardware ports to which you attach cables; rather, they are integers that allow clients to request different services on the same server.] The port number specifies where a server waits for and receives connections from clients -- this is frequently called the handshake point. ... Port numbers are positive integers with values up to 65535. Many operating systems reserve port numbers below 1024 for system services (such as email and World Wide Web servers). Generally, these ports should not be specified as connection ports in user programs. In fact, some operating systems require special access privileges to use port numbers below 1024.

With so many ports from which to choose, how does a client know which port to use when requesting a service? You will often hear the term *well-known port number* used

when describing popular services on the Internet such as Web servers and email servers. For example, a Web server waits for clients to make requests at port 80 by default. All Web browsers know this number as the well-known port on a Web server where requests for HTML documents are made. So when you type a URL into a Web browser, the browser normally connects to port 80 on the server. Similarly, the JSDK WebServer uses port 8080 as its well-known port number. ...."

From the above, it is clear that there are typically 65535 communication ports on a personal computer, and that the lower 1K port (i.e. lower 1024 ports) are reserved for standardized operations, such as printing, email, html access, etc. This is supported by text book, "MCSE: Networking Essential, 2<sup>nd</sup> Edition, page 551, (Exhibit B), which offers an explanation of ports reserved as interrupt locations for input/output (i.e., I/O) applications, which states:

**"Ports (I/O Addresses)**

Along with an interrupt, most devices use a small part of the upper memory area to send data back and forth to the CPU. This area is called a port address or an I/O address. These addresses are used in small blocks--usually 16 bytes or fewer. Addresses are specified using a hexadecimal number and typically range from 300 to 360.

As with interrupts, multiple devices using the same address can cause a conflict. The use of addresses is not as standardized as interrupts, so you will probably need to consult a device's manual to see which port address it is using."

The above definition of a port as a software interface for transferring information packets between different applications and different machines is further explained in dictionary, "IEEE 100, The Authoritative Dictionary of IEEE Standard Terms", 7<sup>th</sup> Edition (Exhibit C), which on page 844 defines the term "port" as used in network applications as,

"(5) A source or destination of data transferred by a Data Transfer class command...Within IEEE Std 1149.5-1995, a port is defined by a module address, port ID meaningful to the MTM-Bus interface logic of that module, and the semantics and structure of packets by which data can be conveyed to and/or from that port. This latter often entails some description of the application to/from which data are passed. A port is selected/accessed/addressed via a Data Transfer class command."

The point is that a "port", as used in the present specification and claims is not a generic term for access to an application, but rather is a well established term for well defined memory locations used to transfer information packets in network communications. When a web browser establishes a link with a network server, the two agree on a network port through which information is

sent/received on each machine. In the present invention, the stand-alone Java application monitors, i.e. listens to, the port that the web browser uses to transfer information, and captures any parameters intended for itself. In this way, the web browser can continue to control the Java application (i.e. transmit program parameters) without having any direct influence on the application's execution, permissions, and abilities.

The Office Action appears to equate this use of ports by a stand-alone application to obtain program parameters transmitted between a web browser and a network server, with the Java Native Interface, JNI. The two are very different. The JNI is a special library of methods (i.e. software routines) that permit a Java program to access routines and libraries written in a different software language (typically C or C++). That is, JNI permits a Java program to access routines already existing on the host machine, but written in a different language that is tied to that specific host machine. In this manner, a new Java program can make use of legacy libraries written in an older program without having to rewrite the libraries in the Java language.

The JNI does not require that network ports be accessed, or monitored, to capture program parameters because the entire Java application (that uses JNI) is compiled incorporating the native (foreign language) libraries. In essence from the stand point of the running Java program, there is no major difference between accessing Java-based libraries and native libraries (except perhaps for additional security checks).

Web page "<http://java.sun.com/docs/books/tutorial/native1.1/>", (Exhibit D), provided by Sun Microsystems Inc., the makers and maintainers of the Java language, explained that,

"The JNI is for programmers who must take advantage of platform-specific functionality outside of the Java Virtual Machine. Because of this, it is recommended that only experienced programmers should attempt to write native methods or use the Invocation API!"

And in "<http://java.sun.com/docs/books/tutorial/native1.1/concepts/index.html>" (Exhibit E) also provided by Sun Microsystems, it states,

"The JNI allows Java code that runs within a Java Virtual Machine (VM) to operate with applications and libraries written in other languages,

such as C, C++, and assembly. In addition, the *Invocation API* allows you to embed the Java Virtual Machine into your native applications.

It must be noted that since JNI is essentially a tool to permit Java programs to access libraries written in "platform-specific" languages, Java programs that make use of JNI lose their portability between computers of different platforms (i.e. operating systems). Thus, JNI changes the character of a Java program in a very real and practical sense. That is, a Java program uses the Java virtual machine to translate Java instructions into platform specific instructions. Therefore, one needs a different Java virtual machine, JVM, for each platform. Native libraries, however, are tied to a specific platform and are compiled into platform-specific machine code, not into instructions that can be interpreted the JVM. Thus, the Java program that uses the JNI ceases to be portable, which makes it less useful for internet applications where platform-free applications are most desirable. This is explained more fully in book "The Java Native Interface" by Sheng Liang, © 1999 Sun Microsystems, Inc., pages 3-9 (Exhibit F), wherein it states that,

"...Applications that use the JNI can incorporate native code written in programming languages such as C and C++, as well as code written in the Java programming language.

... 1.1 The Java Platform and Host Environment

The Java platform is a programming environment consisting of the Java virtual machine (VM) and the Java Application Programming Interface (API). Java applications are written in the Java programming language, and compiled into a machine-independent binary class format. A class can be executed on any Java virtual machine implementation. The Java API consists of a set of predefined classes. Any implementation of the Java platform is guaranteed to support the Java programming language, virtual machine, and API.

The term *host environment* represents the host operating system, a set of native libraries, and the CPU instruction set. Native applications are written in native programming languages such as C and C++, compiled into host-specific binary code, and linked with native libraries. Native applications and native libraries are typically dependent on a particular host environment. A C application built for one operating system, for example, typically does not work on other operating systems."

...1.3 Implications of Using the JNI

Remember that once an application uses the JNI, it risks losing two benefits of the Java platform.

First, Java applications that depend on the JNI can no longer readily run on multiple host environments. ..."

Returning now to the Office Action rejections, claims 1, 20, and 11 were rejected in view of Spyker, column 15, lines 12-20, which the Office asserts shows "passing parameters to a port, the port in communication with the application,

wherein the application is configured to access native libraries of the archive file". Applicants respectfully disagree and point out that Spyker, column 15, lines 12-20 states,

" At Block 745, a platform-dependent JNI (Java Native Interface) is invoked. As is known in the art, the JNI is a standard, virtual machine independent interface used to enable Java applications to call native libraries of code written in other languages such as C or C++. The appropriate environment variables and application parameters are passed on this invocation, enabling Block 750 to finalize the setting of environment variables and then start the system process with the application program executing within it."

As explained above, the JNI is inherently different from the present invention. While JNI permits a Java program to call foreign language libraries using the typical format of embedding parameters within a method call, the present invention is a stand-alone application that monitors a port used by a web browser to capture parameters intended for itself. By doing this, the Java program of present invention can be running on a host machine, maintains its autonomy from a web browser on the same machine, while still maintain a communication link with autonomous web browser.

It should further be noted that Spyker et al. describe the use of Java applets, not stand-alone applications. As is also explained above, applets are inherently different and more restricted than applications. This confusion between applets and applications may be due to Spyker et al.'s purposeful, and erroneous, use the term application when they intend applet. Spyker et al. explain this practice in col. 8, lines 6-9, wherein they state,

"The present invention also enables applets to be run without use of a browser, as if the applet was an application (and therefore all executable programs will be referred to hereinafter as "applications")."

As is explained above, whether an applet is run within a web browser, or in a separate window making it appear as a separate application, it is still an applet subject to the same limitations as the web browser.

The relevant use of "port" was moved from claim 11 to its base claim 9.

Applicants further note that claim 15 also include a recitation stating, "the

application further configured to listen to a port such that any of the options selected by a user are transmitted to the application by a control module through the port". Thus, Applicants respectfully contend that this use of a network port by a stand-alone Java application, as recited in independent claims 1, 9, 15, and 20 is not taught or suggested by the cited prior art.

In reference to claims 1, 21, and 15, the Office Action asserts that Spyker, column 7, line 53 through column 8, line 3, describe that "wherein passing parameters to a port further includes: configuring the application to listen to the port; and sending the parameter over the port, the parameter being sent by a control module." Applicants strongly disagree. As explained above, Spyker describes the use of JNI to access foreign language native libraries, which does not use ports. Furthermore, the Spyker, column 7, line 53 through column 8, line 3, states:

"... The server to which the client computer connects may be functioning as a Web server, where that Web server provides services in response to requests from a client connected through the Internet. Alternatively, the server may be in a corporate intranet or extranet of which the client's workstation is a component. The present invention operates independently of the communications protocol used to send messages or files between the client and server,..."

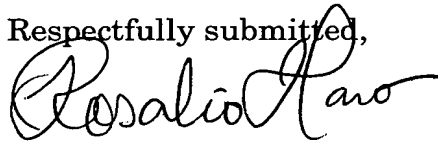
This is in direct conflict with the present invention, which requires that the Java application listen to the port (which is part of the communication protocol used to send messages between a client and a server) to obtain its operating parameters.

In reference to claims 3, 22 and 12, the Office Action states that Spyker, column 7, line 66 through column 8, line 3 shows, "The method as recited in claim 2, wherein the port is a TCP/IP port". As just shown immediately above, Spyker teaches the exact opposite by stating that this invention operates independently of communication protocol used to send messages of files between the client and server. The cited excerpt merely cites TCP/IP as an example of a communication protocol, it does not negate Spyker's assertion that he uses JNI, not ports.

Also in reference to the rejections of claims 4, 23, and 18, the Office Action asserts that Spyker teaches, "wherein the application is a Java application, the Java application configured to be executed by a Java virtual machine of an operating system of the client (Spyker, column 1, lines 22-24)". Applicants respectfully note that cited excerpt is merely a Spyker statement describing the operation of Java applications in the prior art, and not a specific description of his invention. Applicants further note that Spyker's invention is applied to an applet, although he states application in part of his description. This is an erroneous replacement of the term "applet" for "application", as is explained above in reference to Spyker col. 8, lines 6-9. As is also explained above, an applet is inherently different from an application, and the two may not be interchanged.

In view of the foregoing amendments and remarks, Applicants respectfully request favorable reconsideration of the present application.

Respectfully submitted,



Rosalio Haro  
Registration No. 42,633

Please address all correspondence to:

Epson Research and Development, Inc.  
Intellectual Property Department  
150 River Oaks Parkway, Suite 225  
San Jose, CA 95134  
Phone: (408) 952-6000  
Facsimile: (408) 954-9058  
Customer No. 20178

Date: June 3, 2005



---

# JAVA<sup>TM</sup> HOW TO PROGRAM

---

THIRD EDITION

H. M. Deitel  
Deitel & Associates, Inc.

P. J. Deitel  
Deitel & Associates, Inc.

PRENTICE HALL, Upper Saddle River, New Jersey 07458

**Deitel & Deitel**  
**Books and Cyber Classrooms**  
published by  
**Prentice Hall**

**Visual Studio® Series**

*Getting Started with Microsoft® Visual C++™ 6*  
*with an Introduction to MFC*

*Visual Basic® 6 How to Program*

*Getting Started with Microsoft® Visual J++® 1.1*

**How to Program Series**

*Java™ How to Program, 3/E*

*C How to Program, 2/E*

*C++ How to Program, 2/E*

*Visual Basic® 6 How to Program*

**Multimedia Cyber Classroom Series**

*Java™ Multimedia Cyber Classroom, 3/E*

*C & C++ Multimedia Cyber Classroom, 2/E*

*Visual Basic® 6 Multimedia Cyber Classroom*

**The Complete Training Course Series**

*The Complete Java™ Training Course, 3/E*

*The Complete C++ Training Course, 2/E*

*The Complete Visual Basic® 6 Training Course*

For continuing updates on Prentice Hall and Deitel & Associates, Inc. publications  
visit the Prentice Hall web site

<http://www.prenhall.com/deitel>

To communicate with the authors, send email to:

[deitel@deitel.com](mailto:deitel@deitel.com)

For information on corporate on-site seminars and public seminars offered by  
Deitel & Associates, Inc. worldwide, visit:

<http://www.deitel.com>

**J<sup>A</sup>**

**H. M. D**  
Deitel & Assoc

**P. J. De**  
Deitel & Assoc

part of a Web server, please refer to your Web server's documentation on how to install a servlet. For our examples, we demonstrate servlets with the JSDK server.

The JSDK comes with the *JSDK WebServer* so you can test your servlets. The JSDK WebServer assumes that the **.class** files for the servlets are *installed* in the subdirectory

`webpages\WEB-INF\servlets`

of the JSDK install directory on Windows or

`webpages/WEB-INF/servlets`

on UNIX. To install a servlet, first compile the servlet with **javac** as you normally would any other Java source code file. Next, place the **.class** file containing the compiled servlet class in the **servlets** directory. This installs the servlet on the JSDK WebServer.

In the JSDK install directory are a Windows batch file (**startserver.bat**) and a UNIX shell script (**startserver**) that can be used to start the JSDK WebServer on Windows and UNIX, respectively. [Note: The JSDK also provides **stopserver.bat** and **stopserver** to terminate the JSDK WebServer on Windows and UNIX, respectively.] Type the appropriate command for your platform in a command window. When the server starts executing it displays the following command line output:

```
JSDK WebServer Version 2.1
Loaded configuration from file:D:\jsdk2.1/default.cfg
endpoint created: :8080
```

indicating that the JSDK WebServer is waiting for requests on this computer's *port number* 8080. [Note: Ports in this case are not physical hardware ports to which you attach cables; rather, they are integers that allow clients to request different services on the same server.] The port number specifies where a server waits for and receives connections from clients—this is frequently called the *handshake point*. When a client connects to a server to request a service, the client must specify the proper port number; otherwise, the client request cannot be processed. Port numbers are positive integers with values up to 65535. Many operating systems reserve port numbers below 1024 for system services (such as email and World Wide Web servers). Generally, these ports should not be specified as connection ports in user programs. In fact, some operating systems require special access privileges to use port numbers below 1024.

With so many ports from which to choose, how does a client know which port to use when requesting a service? You will often hear the term *well-known port number* used when describing popular services on the Internet such as Web servers and email servers. For example, a Web server waits for clients to make requests at port 80 by default. All Web browsers know this number as the well-known port on a Web server where requests for HTML documents are made. So when you type a URL into a Web browser, the browser normally connects to port 80 on the server. Similarly, the JSDK WebServer uses port 8080 as its well-known port number. You can specify a different port for the JSDK WebServer by editing the file **default.cfg** in the JSDK install directory. Change the line

```
server.port=8080
```

to specify the port on which you would like the JSDK WebServer to await requests.

**MCSE:**

**EXHIBIT B**

# **NETWORKING ESSENTIALS**

## **Study Guide**

**SECOND  
EDITION**

**THE  
#1  
SELLERS:  
NETWORK PRESS  
STUDY GUIDES IN PRINT**



**EXAM 70-058**

**JAMES CHELLIS  
CHARLES PERKINS  
MATTHEW STREBE**

**EVERYTHING YOU NEED TO KNOW TO PASS  
THE NETWORKING ESSENTIALS EXAM**

**PREPARE FOR MICROSOFT'S NEW ADAPTIVE  
AND SIMULATIVE EXAMS WITH HUNDREDS OF  
HANDS-ON EXERCISES AND CHALLENGING  
PRACTICE QUESTIONS**



**INCLUDED ON THE CDS:**

- **LEARNKEY'S NETWORKING ESSENTIALS  
#1 CD-ROM—LIVE VIDEO INSTRUCTION**
- **MICROSOFT'S TRAIN\_CERT OFFLINE UPDATE**
- **THE NETWORK PRESS EDGE TEST FOR  
NETWORKING ESSENTIALS**



**Microsoft Certified**  
**Professional**  
*Approved Study Guide*

Associate Publisher: Guy Hart-Davis  
Contracts and Licensing Manager: Kristine Plachy  
Acquisitions & Developmental Editor: Neil Edde  
Editor: Ben Miller  
Technical Editor: Maryann Brown  
Book Designer: Patrick Dintino  
Graphic Illustrator: Inbar Berman  
Desktop Publisher: Maureen Fors  
Production Coordinator: Eryn Osterhaus  
Indexer: Hugh Maddocks  
Companion CD: Molly Sharp and John D. Wright  
Cover Designer: Archer Design  
Cover Illustrator/Photographer: The Image Bank

Screen reproductions produced with Collage Complete.

Collage Complete is a trademark of Inner Media Inc.

SYBEX, Network Press, and the Network Press logo are registered trademarks of SYBEX Inc.

**TRADEMARKS:** SYBEX has attempted throughout this book to distinguish proprietary trademarks from descriptive terms by following the capitalization style used by the manufacturer.

The author and publisher have made their best efforts to prepare this book, and the content is based upon final release software whenever possible. Portions of the manuscript may be based upon pre-release versions supplied by software manufacturer(s). The author and the publisher make no representation or warranties of any kind with regard to the completeness or accuracy of the contents herein and accept no liability of any kind including but not limited to performance, merchantability, fitness for any particular purpose, or any losses or damages of any kind caused or alleged to be caused directly or indirectly from this book.

SYBEX is an independent entity from Microsoft Corporation, and not affiliated with Microsoft Corporation in any manner. This publication may be used in assisting students to prepare for Microsoft Certified Professional Exam. Neither Microsoft Corporation, its designated review company, nor SYBEX warrants that use of this publication will ensure passing the relevant exam. Microsoft is either a registered trademark or trademark of Microsoft Corporation in the United States and/or other countries.

First edition copyright ©1997 SYBEX Inc.  
Copyright ©1998 SYBEX Inc., 1151 Marina Village Parkway, Alameda, CA 94501. World rights reserved. No part of this publication may be stored in a retrieval system, transmitted, or reproduced in any way, including but not limited to photocopy, photograph, magnetic or other recording, without the prior agreement and written permission of the publisher.

Library of Congress Card Number: 97-81247  
ISBN: 0-7821-2220-5

Manufactured in the United States of America

10 9 8

When two devices use the same interrupt, a conflict occurs. This conflict may cause the computer to immediately crash (refuse to function at all), or it may allow things to work normally and then crash when one or both of the devices are used. In any case, you should resolve the conflict quickly. It can seriously affect the operation of your computer, and you may lose precious data.

### Ports (I/O Addresses)

Along with an interrupt, most devices use a small part of the upper memory area to send data back and forth to the CPU. This area is called a port address or an I/O address. These addresses are used in small blocks—usually 16 bytes or fewer. Addresses are specified using a hexadecimal number and typically range from 300 to 360.

As with interrupts, multiple devices using the same address can cause a conflict. The use of addresses is not as standardized as interrupts, so you will probably need to consult a device's manual to see which port address it is using.

### Memory Addresses

Some devices, particularly video cards and high-speed network cards, use another area of memory as a larger buffer. These areas are specified with a hexadecimal number and usually range from C000 to E000.

### DMA Channels

Another resource a device may use is a direct memory access (DMA) channel. These are high-speed interfaces to the bus that allow a device to access memory directly. DMA channels are numbered from 0 to 2 and are typically used only by time-critical devices, such as sound cards and high-speed disk controllers.

## The Data Bus

A CPU by itself is useless. It can process data but can't read it from a disk or display it on your screen. For the CPU to communicate with the memory and output devices, there must be a medium through which data is transmitted. On a PC motherboard, this medium is known as the bus. The bus serves as a channel through which data is transmitted. The bus that connects the CPU to the memory on a motherboard is known as the data bus. It is made of printed-circuit wiring on the motherboard. This bus also connects to the expansion

**EXHIBIT   C**

**IEEE 100**  
**The Authoritative Dictionary of**  
**IEEE Standards Terms**

**Seventh Edition**



**IEEE**

Published by  
Standards Information Network  
IEEE Press

Trademarks and disclaimers

*IEEE believes the information in this publication is accurate as of its publication date; such information is subject to change without notice. IEEE is not responsible for any inadvertent errors.*

*Other tradenames and trademarks in this document are those of their respective owners.*

*The Institute of Electrical and Electronics Engineering, Inc.  
3 Park Avenue, New York, NY, 10016-5997, USA*

*Copyright © 2000 by the Institute of Electrical and Electronics Engineers, Inc. All rights reserved. Published December 2000. Printed in the United States of America.*

*No part of this publication may be reproduced in any form, in an electronic retrieval system or otherwise, without the prior written permission of the publisher.*

*To order IEEE Press publications, call 1-800-678-IEEE.*

*Print: ISBN 0-7381-2601-2*

*SP1122*

*See other standards and standards-related product listings at: <http://standards.ieee.org/>*

*The publisher believes that the information and guidance given in this work serve as an enhancement to users, all parties must rely upon their own skill and judgement when making use of it. The publisher does not assume any liability to anyone for any loss or damage caused by any error or omission in the work, whether such error or omission is the result of negligence or any other cause. Any and all such liability is disclaimed.*

*This work is published with the understanding that the IEEE is supplying information through this publication, not attempting to render engineering or other professional services. If such services are required, the assistance of an appropriate professional should be sought. The IEEE is not responsible for the statements and opinions advanced in this publication.*

Library of Congress Cataloging-in-Publication Data

IEEE 100 : the authoritative dictionary of IEEE standards terms.—7th ed.

p. cm.

ISBN 0-7381-2601-2 (paperback : alk. paper)

1. Electric engineering—Dictionaries. 2. Electronics—Dictionaries. 3. Computer engineering—Dictionaries. 4. Electric engineering—Acronyms. 5. Electronics—Acronyms. 6. Computer engineering—Acronyms. I. Institute of Electrical and Electronics Engineers.

TK9 .J28 2000  
621.3'03—dc21

00-050601



terms of this guide, the hose is used to carry water.) (T&D/PE) 957-1995

**polyplexer** Equipment combining the functions of duplexing and lobe switching. (AES/RS) 686-1990

**poly-sol** Plastic additive used in some washing applications to break down surface adhesion. (T&D/PE) 957-1987s

**polyvinyl chloride** An insulator in cable coatings and coaxial cable foam compositions. (C) 610.7-1995

**pondage (power operations)** Hydroreserve and limited storage capacity that provides only daily or weekly regulation of streamflow. (PE/PSE) 858-1987s

**pondage station** A hydroelectric generating station with storage sufficient only for daily or weekend regulation of flow. *See also:* generating station. (T&D/PE) [10]

**p-on-n solar cells (photovoltaic power system)** Photovoltaic energy-conversion cells in which a base of n-type silicon (having fixed positive holes in a silicon lattice and electrons that are free to move) is overlaid with a surface layer of p-type silicon (having fixed electrons in a silicon lattice and positive holes that are free to move). (AES) [41]

**pool cathode** A cathode at which the principal source of electron emission is a cathode spot on a metallic pool electrode. (ED) [45]

**pool-cathode mercury-arc converter** A frequency converter using a mercury-arc pool-type discharge device. (IA) 54-1955w, 169-1955w

**pool rectifier** A gas-filled rectifier with a pool cathode, usually mercury. (ED) [45], [84]

**pool tube** A gas tube with a pool cathode. *See also:* electronic controller. (ED) [45]

**POP** *See:* point of presence.

**pop** *See:* pull.

**populate** *See:* load.

**population (1) (data management)** The number of records in a file or database. (C) 610.5-1990w

**(2) (utility power systems)** Transformers that have given common specific characteristics. (PE/TR) C57.117-1986r

**population, conceptual** *See:* conceptual population.

**population inversion (laser maser)** A nonequilibrium condition of a system of weakly interacting particles (electronics, atoms, molecules, or ions) which exists when more than one-half of the particles occupy the higher of two energy states. (LEO) 586-1980w

**pop-up menu (1)** A menu that is brought into view as a result of a selection action other than choosing a menu-bar label. *Contrast:* pull-down menu. (PE/NP) 1289-1998

**(2)** A menu that appears outside of menu bar when requested, usually as the result of pressing BMenu or KMenu. (C) 1295-1993w

**pores (electroplating)** Micro discontinuities in a metal coating that extend through to the base metal or underlying coating. *See also:* electroplating. (LEO) 586-1980w

**Port A** Port Object. Context may indicate that the Port Object is of a specific class. (IM/ST) 1451.1-1999

**port (1) (electronic devices or networks)** A place of access to a device or network where energy may be supplied or withdrawn or where the device or network variables may be observed or measured. *Notes:* 1. In any particular case, the ports are determined by the way the device is used and not by its structure alone. 2. The terminal pair is a special case of a port. 3. In the case of a waveguide or transmission line, a port is characterized by a specified mode of propagation and a specified reference plane. 4. At each place of access, a separate port is assigned to each significant independent mode of propagation. 5. In frequency changing systems, a separate port is also assigned to each significant independent frequency response. *See also:* network analysis; optoelectronic device; waveguide. (ED/IM/HFIM) [46], [45], [40]

**(2) (rotating machinery)** An opening for the intake or discharge of ventilating air. (PE) [9]

**(3) (rotating machinery)** (for a waveguide component) A means of access characterized by a specified reference plane and a specified propagating mode in a waveguide which permits power to be coupled into or out of a waveguide component. *Note:* At low frequencies the port is synonymous with a terminal pair. 2. To each propagating mode at a specified reference plane there corresponds a distinct port. (MTT) 146-1980w

**(4) (broadband local area networks)** An electrical interface that has defined operating boundaries. The specific references within IEEE Std 802.7-1989 assume ports to be 75  $\Omega$  transmission line interfaces that have an associated connector to which the signals pass. (LM/C) 802.7-1989r

**(5)** A source or destination of data transferred by a Data Transfer class command into and/or out of an S-module. A port may be an on-module memory, on-module interface, a peripheral attached to a module, or some other mechanism to/from which data is passed. Within IEEE Std 1149.5-1995, a port is defined by a module address, a port ID meaningful to the MTM-Bus interface logic of that module, and the semantics and structure of packets by which data can be conveyed to and/or from that port. This latter often entails some description of the application to/from which data are passed. A port is selected/accessed/addressed via a Data Transfer class command. (TT/C) 1149.5-1995

**(6)** The physical interconnection point or an access point for a communication link. (C) 610.7-1995

**(7)** An input or output connection between a peripheral device and a computer. *See also:* parallel port; serial port; mouse port; input-output port. (C) 610.10-1994w

**(8)** A physical layer entity in a node that connects to either a cable or backplane and provides one end of a physical connection with another node. (C/MM) 1394-1995

**(9)** A signal interface provided by token ring stations, passive concentrator lobes, active concentrator lobes, or concentrator trunks that is generally terminated at a media interface connector (MIC). Ports may or may not provide physical containment of channels. *See also:* Bridge Port. (C/LM/C/LM) 802.1G-1996, 8802-5-1998

**(10)** An interface point connecting a communications channel and a device. (PE/SUB) 1379-1997

**(11)** A segment or Inter-Repeater Link (IRL) interface of a repeater unit. (C/LM) 802.3-1998

**(12)** A conceptual point at which a cell or a hierarchical design unit makes its interface available to higher levels in the design hierarchy. (C/DA) 1481-1999

**(13)** An abstraction of an access point to network communications. (IM/ST) 1451.1-1999

**(14)** The part of the physical layer (PHY) that allows connection to one other node. (C/MM) 1394a-2000

**(15)** A physical entity that allows import or export of one or more cartridges from a library. (C/SS) 1244.1-2000

**(16)** *See also:* link interface. (C/BA) 1355-1995

**(17)** *See also:* Bridge Port. (C/LM) 802.1G-1996

**portability (1) (software)** The ease with which a system or component can be transferred from one hardware or software environment to another. *Synonym:* transportability. *See also:* machine independent. (C) 610.12-1990

**(2) (application software)** The ease with which application software and data can be transferred from one application platform to another. (C/PA) 14252-1996

**(3)** The capability of being moved between differing environments without losing the ability to be applied or processed. (ATLAS) 1232-1995

**(4)** The capability of being read and/or interpreted by multiple systems. (SCC20) 1232.1-1997

**(5)** The ease with which software can be transferred from one system or environment to another. A relative measure of effort, inversely proportional to the level of modification required for software to be transferred from one system or environment to another. (SCC20) 1226-1998

**portable (x-ray)** X-ray equipment designed to be hand-carried. (NEC/NESC) [86]

**portable appliance** An appliance that can easily be moved from one location to another. For the purpose of this article, other than built-in are coolers, refrigerators, gas range washers without booster. *See also:* appliance.

**portable battery** A storage device for power. *See also:* transportation. *See also:* battery.

**portable character set** A character set that is supported on all computers that are trusted against the small character set.

**portable character string** A character string that is supported on all computers that are trusted against the small character set. *See also:* portable character set. *See also:* cataloged, all such.

**portable computer** A personal computer configured to permit transport. *Note:* U.S. Federal Government "portable" to objects we use. *See also:* notebook computer; laptop computer.

**portable concentric mine** A mine in which one conductor located at the center and the other conductor strands located concentrically around the center conductor. *See also:* concentric mine.

**portable filename character** A character which portable filename which is portable across configurations. *See also:* portable filename. *See also:* portable filename. *See also:* portable filename.

**portable filename character** A character which portable filename which is portable across configurations. *See also:* portable filename. *See also:* portable filename. *See also:* portable filename.

**portable filename character** A character which portable filename which is portable across configurations. *See also:* portable filename. *See also:* portable filename. *See also:* portable filename.

**portable identifier character** A character which portable identifier which is portable across configurations. *See also:* portable identifier. *See also:* portable identifier. *See also:* portable identifier.

**portable lighting (illumination)** Lighting equipment designed to be hand-carried.

**portable luminaire (illumination)** A luminaire which is not permanent.

**portable mine blower** A blower used for ventilation into main ventilating system spaces through a duct.

**portable mine cable** A cable used for mobile or stationary electric energy when practicable. *See also:* portable mine cable.

**portable mining-type reformer** A reformer that is suitable for use in the restrictive areas of a mine.





## Trail: Java Native Interface


by *Beth Stearns*


The lessons in this trail show you how to integrate native code with programs written in Java. You will learn how to write *native methods*. Native methods are methods implemented in another programming language such as C.


**The JNI is for programmers who must take advantage of platform-specific functionality outside of the Java Virtual Machine. Because of this, it is recommended that only experienced programmers should attempt to write native methods or use the Invocation API!**


 **Overview of the JNI** begins with an introduction to the JNI concepts.

 **Writing Java Programs with Native Methods** describes how to compile and run a Java program with a native method. It walks you step by step through a simple example (the "Hello World!" of native methods) to illustrate how to write, compile, and run a Java program that includes native methods.

 **Integrating Java and Native Programs** shows you how to map Java types to native types. This lesson includes information about passing arguments of various data types into a native method and returning values of various data types from a native method. It also shows how to implement a native method within a Java program.

 **Interacting with Java from the Native Side** describes many useful functions that your native language code can use to access Java objects and their members, create Java objects, throw exceptions, and more.

 **Invoking the Java Virtual Machine** explains how to invoke the Java Virtual Machine from your native application.

 **Summary of the JNI** lists the JNI methods and mapping tables to remind you of what you've learned.

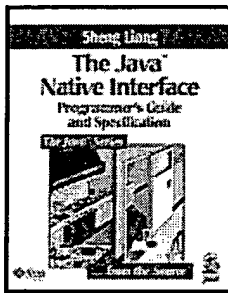
---

### Note:

Available now from [amazon.com](http://amazon.com)◆

For more complete information, refer to the new book, *The Java™ Native Interface: Programmer's Guide and Specification* written by Sheng Liang. Published by Addison Wesley Longman, Inc. June 1999.

This book is the definitive resource and a comprehensive guide to working with the JNI. Entirely up-to-date, the book offers a tutorial,



a detailed description of JNI features and programming techniques, JNI design justifications, and the official specification for all JNI types and functions.

---

**Security consideration:** Note that the ability to load dynamic libraries is subject to approval by the current security manager. When working with native methods, you must load dynamic libraries. Some applets may not be able to use native methods because the browser or viewer they are running in restricts the ability to load dynamic libraries. See [Security Restrictions](#)◆ for information about the security restrictions placed on applets.

---

**Note:** MacOS programmers should refer to [MacOS Runtime for Java](#)◆.

---

Programmers interested in writing native methods in releases prior to 1.1 can [download](#) the old version of this trail. It describes native methods for the 1.0.2 release of the JDK.



[Start of Tutorial](#)

[Search](#)  
[Feedback Form](#)

[Copyright](#) 1995-2005 Sun Microsystems, Inc. All rights reserved.



---

Trail: Java Native Interface

## Lesson: Overview of the JNI

The Java Native Interface (JNI) is the native programming interface for Java that is part of the JDK. By writing programs using the JNI, you ensure that your code is completely portable across all platforms.

The JNI allows Java code that runs within a Java Virtual Machine (VM) to operate with applications and libraries written in other languages, such as C, C++, and assembly. In addition, the *Invocation API* allows you to embed the Java Virtual Machine into your native applications.

Programmers use the JNI to write native methods to handle those situations when an application cannot be written entirely in the Java programming language. For example, you may need to use native methods and the JNI in the following situations:

- The standard Java class library may not support the platform-dependent features needed by your application.
- You may already have a library or application written in another programming language and you wish to make it accessible to Java applications.
- You may want to implement a small portion of time-critical code in a lower-level programming language, such as assembly, and then have your Java application call these functions.

Programming through the JNI framework lets you use native methods to do many operations. Native methods may represent legacy applications or they may be written explicitly to solve a problem that is best handled outside of the Java programming environment.

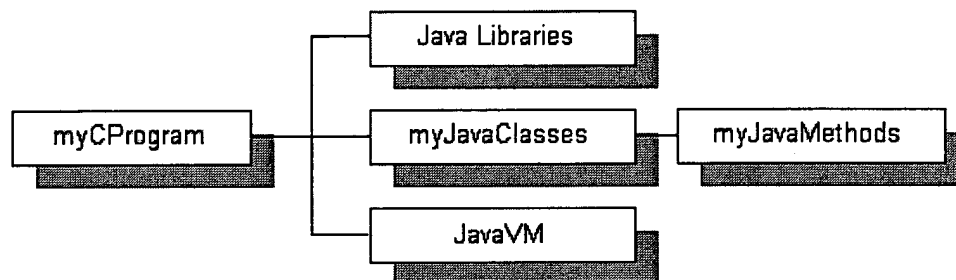
The JNI framework lets your native method utilize Java objects in the same way that Java code uses these objects. A native method can create Java objects, including arrays and strings, and then inspect and use these objects to perform its tasks. A native method can also inspect and use objects created by Java application code. A native method can even update Java objects that it created or that were passed to it, and these updated objects are available to the Java application. Thus, both the native language side and the Java side of an application can create, update, and access Java objects and then share these objects between them.

Native methods can also easily call Java methods. Often, you will already have developed a library of Java methods. Your native method does not need to "re-invent the wheel" to perform functionality already incorporated in existing Java methods. The native method, using the JNI framework, can call the existing Java method, pass it the required parameters, and get the results back when the method completes.

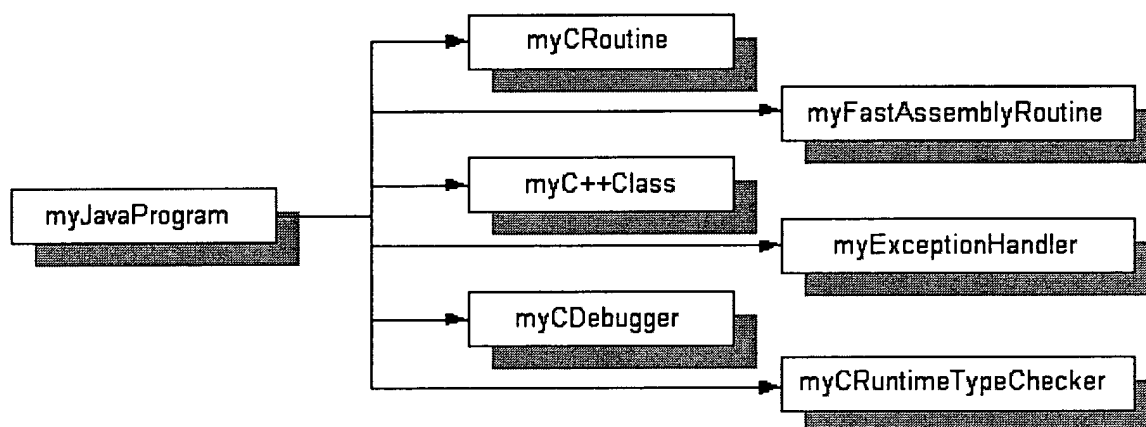
The JNI enables you to use the advantages of the Java programming language from your

native method. In particular, you can catch and throw exceptions from the native method and have these exceptions handled in the Java application. Native methods can also get information about Java classes. By calling special JNI functions, native methods can load Java classes and obtain class information. Finally, native methods can use the JNI to perform runtime type checking.

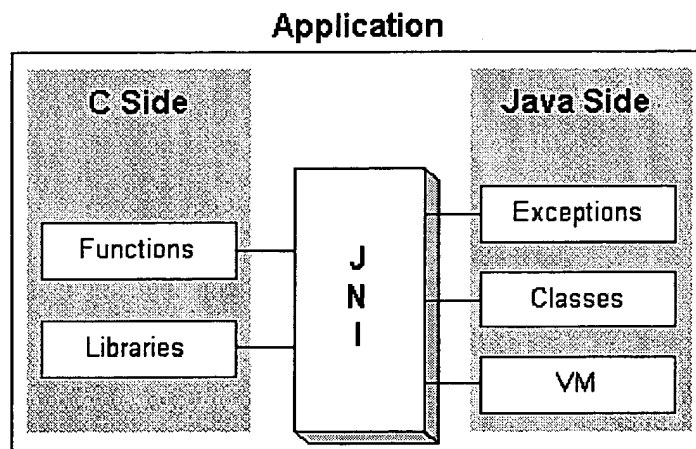
For example, the following figure shows how a legacy C program can use the JNI to link with Java libraries, call Java methods, use Java classes, and so on.



The next figure illustrates calling native language functions from a Java application. This diagram shows the many possibilities for utilizing the JNI from a Java program, including calling C routines, using C++ classes, calling assembler routines, and so on.



It is easy to see that the JNI serves as the glue between Java and native applications. The following diagram shows how the JNI ties the C side of an application to the Java side.





[Start of Tutorial](#) > [Start of Trail](#)

[Search](#)  
[Feedback Form](#)

Copyright 1995-2005 Sun Microsystems, Inc. All rights reserved.

# *The Java™ Native Interface*

## *Programmer's Guide and Specification*

Sheng Liang

[www.javaseries.com](http://www.javaseries.com)

, Second Edition  
Peursem, Jyn

Java™ 2 Platform,

el Deering  
nd Edition

es: Advanced Topics

nstructing GUIs  
nell, Graham Hamilton.

Second Edition:  
1 2 Platform

egies and Tactics

Arnold  
and Practice

dition  
n Haase, Eric Jendrock,

ise Partners  
Building Business  
form, Enterprise Edition  
nul Sharma, Joseph Fialli,

orial and Reference:  
orm

ark Johnson, Enterprise Team  
ns with the Java™ 2

n, Linda DeMichiel, Beth

5™ 2.1, Second Edition:  
ut for the J2EE™ Platform  
lada Matena, James Davidson,  
y Cable, Enterprise Team  
: Edition: Platform and

Tony Ng  
ire and Enterprise Application



**ADDISON-WESLEY**

**An imprint of Addison Wesley Longman, Inc.**

Reading, Massachusetts • Harlow, England • Menlo Park, California  
Berkeley, California • Don Mills, Ontario • Sydney  
Bonn • Amsterdam • Tokyo • Mexico City

Copyright © 1999 Sun Microsystems, Inc.  
901 San Antonio Road, Palo Alto, CA 94303 USA.  
All rights reserved.

Duke™ designed by Joe Palrang.

Sun Microsystems, Inc. has intellectual property rights relating to implementations of the technology described in this publication. In particular, and without limitation, these intellectual property rights may include one or more U.S. patents, foreign patents, or pending applications. Sun, Sun Microsystems, the Sun logo, and all Sun, Java, Jini, and Solaris based trademarks and logos are trademarks or registered trademarks of Sun Microsystems, Inc. in the United States and other countries. UNIX is a registered trademark in the United States and other countries, exclusively licensed through X/Open Company, Ltd.

Sun Microsystems, Inc. (SUN) hereby grants to you a fully paid, nonexclusive, nontransferable, perpetual, worldwide limited license (without the right to sublicense) under SUN's intellectual property rights that are essential to practice this specification. This license allows and is limited to the creation and distribution of clean room implementations of this specification that: (i) include a complete implementation of the current version of this specification without subsetting or supersetting; (ii) implement all the required interfaces and functionality of the Java™ 2 Platform, Standard Edition, as defined by SUN, without subsetting or supersetting; (iii) do not add any additional packages, classes, or interfaces to the java.\* or javax.\* packages or their subpackages; (iv) pass all test suites relating to the most recent published version of the specification of the Java™ 2 Platform, Standard Edition, that are available from SUN six (6) months prior to any beta release of the clean room implementation or upgrade thereto; (v) do not derive from SUN source code or binary materials; and (vi) do not include any SUN source code or binary materials without an appropriate and separate license from SUN.

**U.S. GOVERNMENT USE:** This specification relates to commercial items, processes, or software. Accordingly, use by the United States Government is subject to these terms and conditions, consistent with FAR 12.211 and 12.212.

**THIS PUBLICATION IS PROVIDED "AS IS" WITHOUT WARRANTY OF ANY KIND, EITHER EXPRESS OR IMPLIED, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF MERCHANTABILITY, FITNESS FOR A PARTICULAR PURPOSE, OR NON-INFRINGEMENT.**

**THIS PUBLICATION COULD INCLUDE TECHNICAL INACCURACIES OR TYPOGRAPHICAL ERRORS. CHANGES ARE PERIODICALLY ADDED TO THE INFORMATION HEREIN; THESE CHANGES WILL BE INCORPORATED IN NEW EDITIONS OF THE PUBLICATION. SUN MICROSYSTEMS, INC. MAY MAKE IMPROVEMENTS AND/OR CHANGES IN ANY TECHNOLOGY, PRODUCT, OR PROGRAM DESCRIBED IN THIS PUBLICATION AT ANY TIME.**

The publisher offers discounts on this book when ordered in quantity for special sales. For more information, please contact the Corporate, Government, and Special Sales Group, CEPUB, Addison Wesley Longman, Inc., One Jacob Way, Reading, Massachusetts 01867.

ISBN 0-201-32577-2

4 5 6 7 8 9-MA-0302010099

First printing, June 1999



# Introduction

**T**HE Java™ Native Interface (JNI) is a powerful feature of the Java platform. Applications that use the JNI can incorporate *native code* written in programming languages such as C and C++, as well as code written in the Java programming language. The JNI allows programmers to take advantage of the power of the Java platform, without having to abandon their investments in legacy code. Because the JNI is a part of the Java platform, programmers can address interoperability issues once, and expect their solution to work with all implementations of the Java platform.

This book is both a programming guide and a reference manual for the JNI. The book consists of three parts:

- Chapter 2 introduces the JNI through a simple example. It is a tutorial intended for the beginning users who are unfamiliar with the JNI.
- Chapters 3 to 10 constitute a programmer's guide that gives a broad overview of a number of JNI features. We will go through a series of short but descriptive examples to highlight various JNI features and to present the techniques that have proven to be useful in JNI programming.
- Chapters 11 to 13 present the definitive specification for all JNI types and functions. These chapters are also organized to serve as a reference manual.

This book tries to appeal to a wide audience with different needs for the JNI. The tutorial and programming guide are targeted toward beginning programmers, whereas experienced developers and JNI implementors may find the reference sections more useful. The majority of readers will likely be developers who use the JNI to write applications. The term "you" in this book will implicitly denote developers who program with the JNI, as opposed to JNI implementors or end-users of applications written using the JNI.

The book assumes that you have basic knowledge of the Java, C, and C++ programming languages. If not, you may refer to one of the many excellent books that are available: *The Java™ Programming Language, Second Edition*, by Ken Arnold and James Gosling (Addison-Wesley, 1998), *The C Programming Language, Second Edition*, by Brian Kernighan and Dennis Ritchie (Prentice Hall,

1988), and *The C++ Programming Language, Third Edition*, by Bjarne Stroustrup (Addison-Wesley, 1997).

The remainder of this chapter introduces the background, role, and evolution of the JNI.

## 1.1 The Java Platform and Host Environment

Because this book covers applications written in the Java programming language as well as in native (C, C++, etc.) programming languages, let us first clarify the exact scope of the programming environments for these languages.

The Java platform is a programming environment consisting of the Java virtual machine (VM) and the Java Application Programming Interface (API).<sup>1</sup> Java applications are written in the Java programming language, and compiled into a machine-independent binary class format. A class can be executed on any Java virtual machine implementation. The Java API consists of a set of predefined classes. Any implementation of the Java platform is guaranteed to support the Java programming language, virtual machine, and API.

The term *host environment* represents the host operating system, a set of native libraries, and the CPU instruction set. *Native applications* are written in *native programming languages* such as C and C++, compiled into host-specific binary code, and linked with native libraries. Native applications and native libraries are typically dependent on a particular host environment. A C application built for one operating system, for example, typically does not work on other operating systems.

Java platforms are commonly deployed on top of a host environment. For example, the Java Runtime Environment (JRE) is a Sun product that supports the Java platform on existing operating systems such as Solaris and Windows. The Java platform offers a set of features that applications can rely on independent of the underlying host environment.

## 1.2 Role of the JNI

When the Java platform is deployed on top of host environments, it may become desirable or necessary to allow Java applications to work closely with native code written in other languages. Programmers have begun to adopt the Java platform to build applications that were traditionally written in C and C++. Because of the

<sup>1</sup>. As used herein, the phrases "Java virtual machine" or "Java VM" mean a virtual machine for the Java platform. Similarly, the phrase "Java API" means the API for the Java platform.

existing inve  
and C++ cod

The JNI  
platform, but  
tual machine  
applications  
the JNI.

Java applic  
and library

Figure 1.1

The JNI  
applications  
types of nativ

- You can call func methods grammin mented in
- The JNI : tual macl link with use the i Java pro execute c tion.

Bjarne Strou-  
and evolution

ning language  
first clarify the

f the Java vir-  
e (API).<sup>1</sup> Java  
ompile into a  
d on any Java  
of predefined  
to support the

stem, a set of  
are written in  
o host-specific  
d native librar-  
application built  
other operating

ironment. For  
at supports the  
Windows. The  
ndependent of

t may become  
th native code  
va platform to  
because of the

l machine for the  
orm.

existing investment in legacy code, however, Java applications will coexist with C and C++ code for many years to come.

The JNI is a powerful feature that allows you to take advantage of the Java platform, but still utilize code written in other languages. As a part of the Java virtual machine implementation, the JNI is a *two-way* interface that allows Java applications to invoke native code and vice versa. Figure 1.1 illustrates the role of the JNI.

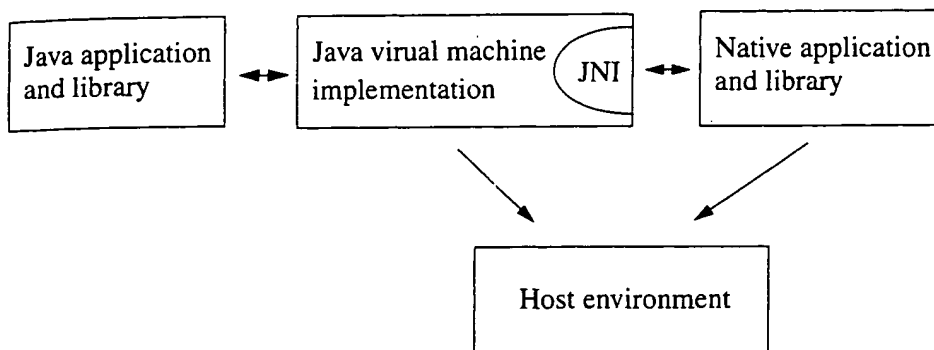


Figure 1.1 Role of the JNI

The JNI is designed to handle situations where you need to combine Java applications with native code. As a two-way interface, the JNI can support two types of native code: native *libraries* and native *applications*.

- You can use the JNI to write *native methods* that allow Java applications to call functions implemented in native libraries. Java applications call native methods in the same way that they call methods implemented in the Java programming language. Behind the scenes, however, native methods are implemented in another language and reside in native libraries.
- The JNI supports an *invocation interface* that allows you to embed a Java virtual machine implementation into native applications. Native applications can link with a native library that implements the Java virtual machine, and then use the invocation interface to execute software components written in the Java programming language. For example, a web browser written in C can execute downloaded applets in an embedded Java virtual machine implementation.

### 1.3 Implications of Using the JNI

Remember that once an application uses the JNI, it risks losing two benefits of the Java platform.

First, Java applications that depend on the JNI can no longer readily run on multiple host environments. Even though the part of an application written in the Java programming language is portable to multiple host environments, it will be necessary to recompile the part of the application written in native programming languages.

Second, while the Java programming language is type-safe and secure, native languages such as C or C++ are not. As a result, you must use extra care when writing applications using the JNI. A misbehaving native method can corrupt the entire application. For this reason Java applications are subject to security checks before invoking JNI features.

As a general rule, you should architect the application so that native methods are defined in as few classes as possible. This entails a cleaner isolation between native code and the rest of the application.

### 1.4 When to Use the JNI

Before you embark on a project using the JNI, it is worth taking a step back to investigate whether there are alternative solutions that are more appropriate. As mentioned in the last section, applications that use the JNI have inherent disadvantages when compared with applications written strictly in the Java programming language. For example, you lose the type-safety guarantee of the Java programming language.

A number of alternative approaches also allow Java applications to interoperate with code written in other languages. For example:

- A Java application may communicate with a native application through a TCP/IP connection or through other inter-process communication (IPC) mechanisms.
- A Java application may connect to a legacy database through the JDBC™ API.
- A Java application may take advantage of distributed object technologies such as the Java IDL API.

A common characteristic of these alternative solutions is that the Java application and native code reside in different processes (and in some cases on different machines). Process separation offers an important benefit. The address space pro-

tection supports native applications to communicate with native code. Sometimes this becomes useful.

- The Java A application operations and inefficiencies.
- You may v for the ove Loading th
- Having an memory fo the same c hosting the cess and lo
- You may v level langu its time in tion of a gr

In summar code that resid

### 1.5 Evolut

The need for . nized since the platform, Java interface that guages such a mentation of java.io, and tures in the un Unfortuna problems:

tection supported by processes enables a high degree of fault isolation—a crashed native application does not immediately terminate the Java application with which it communicates over TCP/IP.

Sometimes, however, you may find it necessary for a Java application to communicate with native code *that resides in the same process*. This is when the JNI becomes useful. Consider, for example, the following scenarios:

- The Java API might not support certain host-dependent features needed by an application. An application may want to perform, for example, special file operations that are not supported by the Java API, yet it is both cumbersome and inefficient to manipulate files through another process.
- You may want to access an existing native library and are not willing to pay for the overhead of copying and transmitting data across different processes. Loading the native library in the same process is much more efficient.
- Having an application span multiple processes could result in unacceptable memory footprint. This is typically true if these processes need to reside on the same client machine. Loading a native library into the existing process hosting the application requires less system resources than starting a new process and loading the library into that process.
- You may want to implement a small portion of time-critical code in a lower-level language, such as assembly. If a 3D-intensive application spends most of its time in graphics rendering, you may find it necessary to write the core portion of a graphics library in assembly code to achieve maximum performance.

In summary, use the JNI if your Java application must interoperate with native code that resides in the same process.

## 1.5 Evolution of the JNI

The need for Java applications to interoperate with native code has been recognized since the very early days of the Java platform. The first release of the Java platform, Java Development Kit (JDK™) release 1.0, included a native method interface that allowed Java applications to call functions written in other languages such as C and C++. Many third-party applications, as well as the implementation of the Java class libraries (including, for example, `java.lang`, `java.io`, and `java.net`), relied on the native method interface to access the features in the underlying host environment.

Unfortunately, the native method interface in JDK release 1.0 had two major problems:

- First, the native code accesses fields in objects as members of C structures. However, the Java virtual machine specification does not define how objects are laid out in memory. If a given Java virtual machine implementation lays out objects in a way other than that assumed by the native method interface, then you have to recompile the native method libraries.
- Second, the native method interface in JDK release 1.0 relies on a conservative garbage collector because native methods can get hold of direct pointers to objects in the virtual machine. Any virtual machine implementation that uses more advanced garbage collection algorithms cannot support the native method interface in JDK release 1.0.

The JNI was designed to overcome these problems. It is an interface that can be supported by all Java virtual machine implementations on a wide variety of host environments. With the JNI:

- Each virtual machine implementor can support a larger body of native code.
- Development tool vendors do not have to deal with different kinds of native method interfaces.
- Most importantly, application programmers are able to write one version of their native code and this version will run on different implementations of the Java virtual machine.

The JNI was first supported in JDK release 1.1. Internally, however, JDK release 1.1 still uses old-style native methods (as in JDK release 1.0) to implement the Java APIs. This is no longer the case in Java 2 SDK release 1.2 (formerly known as JDK release 1.2). Native methods have been rewritten so that they conform to the JNI standard.

The JNI is the native interface supported by all Java virtual machine implementations. From JDK release 1.1 on, you should program to the JNI. The old-style native method interface is still supported in Java 2 SDK release 1.2, but will not (and cannot) be supported in advanced Java virtual machine implementations in the future.

Java 2 SDK release 1.2 contains a number of JNI enhancements. The enhancements are backward compatible. All future evolutions of JNI will maintain complete binary compatibility.

## 1.6 Example Programs

This book contains numerous example programs that demonstrate JNI features. The example programs typically consist of multiple code segments written in the

## INTRODUCTION

Java programming native code refers to how to build JNI code with JDK and Java.

Keep in mind that the JNI is a set of methods or specific code, not on the technology itself. It is defined with JDK and may offer an implementation that you can consult the documentation of your choice.

You can download the latest updates to the

<http://java>

bers of C structures.  
ot define how objects  
implementation lays  
ive method interface,

relies on a conserva-  
old of direct pointers  
implementation that  
it support the native

in interface that can  
n a wide variety of

dy of native code.  
ent kinds of native

rite one version of  
ementations of the

ly, however, JDK  
1.0) to implement  
ase 1.2 (formerly  
so that they con-

l machine imple-  
he JNI. The old-  
ease 1.2, but will  
implementations

ancements. The  
JNI will main-

e JNI features.  
s written in the

Java programming language as well as C or C++ native code. Sometimes the native code refers to host-specific features in Solaris and Win32. We also show how to build JNI programs using the command line tools (such as javah) shipped with JDK and Java 2 SDK releases.

Keep in mind that the use of the JNI is not limited to specific host environments or specific application development tools. The book focuses on writing the code, not on the tools used to build and run the code. The command line tools bundled with JDK and Java 2 SDK releases are rather primitive. Third-party tools may offer an improved way to build applications that use the JNI. We encourage you to consult the JNI-related documentation bundled with the development tools of your choice.

You can download the source code of the examples in this book, as well as the latest updates to this book, from the following web address:

<http://java.sun.com/docs/books/jni/>

**This Page is Inserted by IFW Indexing and Scanning  
Operations and is not part of the Official Record**

**BEST AVAILABLE IMAGES**

Defective images within this document are accurate representations of the original documents submitted by the applicant.

Defects in the images include but are not limited to the items checked:

- ☐ **BLACK BORDERS**
- ☐ **IMAGE CUT OFF AT TOP, BOTTOM OR SIDES**
- ☐ **FADED TEXT OR DRAWING**
- ☐ **BLURRED OR ILLEGIBLE TEXT OR DRAWING**
- ☐ **SKEWED/SLANTED IMAGES**
- ☐ **COLOR OR BLACK AND WHITE PHOTOGRAPHS**
- ☐ **GRAY SCALE DOCUMENTS**
- ☐ **LINES OR MARKS ON ORIGINAL DOCUMENT**
- ☐ **REFERENCE(S) OR EXHIBIT(S) SUBMITTED ARE POOR QUALITY**
- ☐ **OTHER:** \_\_\_\_\_

**IMAGES ARE BEST AVAILABLE COPY.**

**As rescanning these documents will not correct the image problems checked, please do not report these problems to the IFW Image Problem Mailbox.**